CORE JAVA MODULE 3 NOTES

27 June 2025 15:09

➢ Module 3: Control Flow (Conditions & Loops)

No.	Topic
3.1	if, if-else, nested if , if-else-if
3.2	switch-case (String supported)
3.3	for, while, do-while
3.4	break, continue, return
3.5	Interview Questions and Assignment

⇒ Module-3.1 : if, if-else, nested if , if-else-if

Real-Life Analogy:

Imagine your **daily routine**:

- If it's raining, take an umbrella. (if)
- If it's raining, take an umbrella; **otherwise**, wear sunglasses. (if-else)
- If it's a weekday and you're late, **run**; if you're early, **walk**. (nested if)
- If it's Monday, do gym; else if Tuesday, go jogging; else if Wednesday, yoga... (if-else-if ladder)

- **Control flow statements** help us decide **which part of code should execute**, based on **certain conditions**.
  The if, if-else, nested if, and if-else-if are **decision-making statements** that let the program **choose different paths** based on **true/false conditions**.

Syntax + Explanation

1. if Statement

Used when you want to run a block of code **only if** a condition is **true**.

```
if (condition) {
        // runs only if condition is true
        }
```

Example:

```
int marks = 90;
if (marks > 80) {
        System.out.println("Excellent!");
}
```

2. if-else Statement

Used when there are **two possible outcomes** — one for true and one for false.

```
if (condition) {
        // if true
        } else {
        // if false
        }
```

Example:

```
int age = 17;
```

```
if (age >= 18) {
        System.out.println("Eligible to vote");
} else {
        System.out.println("Not eligible");
}
```

3. nested if Statement

An if inside another if.
Used when **one condition depends on another**.

```
if (condition1) {
        if (condition2) {
        // runs only if both are true
        }
   }
```

Example:

```
int age = 20;
String citizen = "Indian";
if (age >= 18) {
        if (citizen.equals("Indian")) {
        System.out.println("Eligible to vote in India");
    }
 }
```

4. if-else-if Ladder

Used when you have **multiple conditions**, and only **one** should execute.

```
if (condition1) {
        // block 1
        } else if (condition2) {
        // block 2
        } else if (condition3) {
        // block 3
        } else {
        // default block
        }
```

Example:

```
int day = 3;
if (day == 1) {
        System.out.println("Monday");
} else if (day == 2) {
        System.out.println("Tuesday");
} else if (day == 3) {
        System.out.println("Wednesday");
} else {
        System.out.println("Invalid day");
}
```

Practice Questions:

1. Write a program to check if a number is **positive or negative**.
2. Check if a person can **get a driving license** (age $\geq$ 18) and has **ID proof**.
3. Use if-else-if to print grades:

- 90+ → A
- 80-89 → B
- 70-79 → C
- <70 → Fail

⇒ Module : 3.2 – switch-case (String Supported)

- **Definition:**

The switch statement is a **multi-way decision maker**. It lets a variable be tested for **equality** against **multiple values (cases)**. It's cleaner and more readable than using **many if-else-if statements**, especially when you are checking a single variable for **multiple constant values**.
Since Java **7**, switch also supports String, making it even more powerful and flexible!

Real-Life Analogy:

Imagine you're ordering tea from a vending machine:

- Press 1 → Masala Tea
- Press 2 → Green Tea
- Press 3 → Lemon Tea
- Press any other key → Invalid option

Instead of using multiple if-else, you just use a switch.

Syntax + Explanation

```
switch (expression) {
        case value1:
        // code
        break;
        case value2:
        // code
        break;
        ...
default:
        // default block (optional)
        }
```


- expression → Can be int, char, enum, or String (from Java 7 onward).
- break → Used to **exit** the switch. Without break, Java performs **fall-through**.
- default → Executes if **no case matches** (like an else).

CODE :

```
int day = 3;

switch (day) {
        case 1:
        System.out.println("Monday");
        break;
                case 2:
                System.out.println("Tuesday");
        break;
                case 3:
                System.out.println("Wednesday");
        break;
default:
        System.out.println("Invalid day");
```

```
}
```

Practice Questions:

1. Take user input for a **number 1 to 7** and print the day of the week using switch.
2. Use a String input to return:
   - "India" → "Namaste"
   - "USA" → "Hello"
   - "France" → "Bonjour"
3. Build a basic calculator using switch: +, -, *, /

⇒ Module : 3.3 – Loops: for, while, do-while

Definition:

A **loop** in Java is a control structure that lets you **repeat a block of code multiple times**.
Instead of writing the same code again and again, you can loop it until a condition is false.
Java has **3 main loops**: for, while, and do-while.

Real-Life Analogy:

Think of a gym workout:

- You want to do **10 pushups**.
- Will you write "pushup" 10 times on paper? No!
- You repeat it inside a loop.

Loops = *"Do this again and again until I say stop!"*

Loop Types:

1. for Loop

Definition:
* Best when you know **how many times** you want to repeat something.

Syntax:

```
for (initialization; condition; update) {
   // code block
}
```

Example:

```
for (int i = 1; i <= 5; i++) {
      System.out.println("Hello " + i);
}
```

Output:

```
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

2. while Loop

Definition:

- Used when you **don't know in advance** how many times to loop — just loop **as long as condition is true**.

Syntax:

```
while (condition) {
   // code block
}
```

Example:

```
int i = 1;
while (i <= 3) {
        System.out.println("Repeat " + i);
i++;
        }
```

3. do-while Loop

Definition:

- It's like while, but it **runs at least once**, even if the condition is false.

Syntax:

```
do {
   // code block
} while (condition);
```

Example:

```
int i = 1;
do {
        System.out.println("Try " + i);
i++;
        } while (i <= 2);
```

Even if condition is false in the beginning — this loop **executes once**.

⇒ Module 3.4 – break, continue, return

Definition:

- These are **jump statements** in Java.
- They help us to **control the flow** of loops or methods by **interrupting** or **exiting** at certain points.

Statement	What It Does
break	**Exits** from a loop or switch
continue	**Skips** the current iteration and goes to the next
return	**Exits** from a method and optionally **returns a value**

✧ Real-Life Analogy: Teacher Checking Exam Papers

Imagine you are a **teacher checking 10 student exam papers** in a loop.

1. break – "Oh no! A student has written abusive words!"

- You're checking papers one by one.
- But when you reach **paper 5**, you find something **inappropriate**.

- You immediately stop checking and **leave the rest**.

```
for (int i = 1; i <= 10; i++) {
        if (i == 5) {
        System.out.println("Student " + i + " wrote something bad! Stopping check.");
        break;
                }
                System.out.println("Checking paper of Student " + i);
}
```

break = Jitni der sab theek tha, check chalta raha. Kuch galat mila, to full stop.

2. continue – "One paper is blank. Skip it and move on."

- You're checking all papers.
- **Paper 4 is blank** – you just skip it and go to the next one.

```
for (int i = 1; i <= 10; i++) {
        if (i == 4) {
        System.out.println("Student " + i + " submitted blank paper. Skipping...");
        continue;
                }
                System.out.println("Checking paper of Student " + i);
}
```

continue = Yeh paper toh khali hai… agli taraf badho.

3. return – "You're a guest teacher, checking only one student's paper."

- A teacher is called just to check **one student's paper**, not the rest.
- As soon as that paper is done, the teacher says **"My job's done, I'm leaving."**

```
public static void checkStudent(int rollNo) {
    System.out.println("Checking paper of Student " + rollNo);
    return;
    // Below code will never execute
    // System.out.println("Thank you for checking!");
}
```

return = Kaam khatam, main nikalta hoon.

⇒ Module 3.5 Beginner-Level Questions & Answers – Control Flow:

No.	Question	Answer
1	w What is the purpose of control flow statements in Java?	w To control the order in which statements are executed based on conditions or repetitions.
2	w What is the difference between if and if-else?	w if runs a block when a condition is true, if-else runs one block if true, another if false.
3	w Can if-else be nested inside another if block?	w Yes, this is called a nested if structure.
4	w When do we use switch over if-else?	w When checking one variable against many constant values (like menu options).
5	w Can switch work with String in Java?	w Yes, from Java 7 onwards.
6	w What is a loop in Java?	w A structure that repeats a block of code until a condition becomes false.
7	w What's the difference between while and do-while loop?	w while checks the condition first, do-while runs once before checking.
8	w What's the best loop to use when the number of repetitions is known?	w for loop.

9	◆w What does break do in a loop?	◆w It exits the loop immediately.
10	◆w What does continue do in a loop?	◆w It skips the current iteration and goes to the next.
11	◆w What does return do in a method?	◆w It exits the method and optionally returns a value.
12	◆w Can we use continue in a switch block?	◆w No, continue only works in loops, not in switch.

Intermediate-Level Questions & Answers – Control Flow

No.	Question	Answer
1	◆w What happens if you forget break in a switch case?	◆w Java will execute all following cases (fall-through behavior) until a break or end of switch.
2	◆w Can we use expressions inside a switch statement?	◆w No, only constants or final variables are allowed in case labels.
3	◆w Can a loop run forever? How?	◆w Yes, if the condition never becomes false (e.g., while(true), for(;;)).
4	◆w What is the scope of loop variables declared inside a loop?	◆w They are only accessible within the loop block.
5	◆w Can a return statement stop a loop?	◆w Yes, if used inside a loop, return exits the whole method, not just the loop.
6	◆w What's the difference between == and equals() in control flow conditions?	◆w == checks memory reference, equals() checks actual content (useful in if with Strings).
7	◆w Can we write a for loop without all three parts (init, condition, update)?	◆w Yes, all parts are optional. Even for(;;) is valid (infinite loop).
8	◆w What is loop control variable?	◆w A variable used to control how many times the loop will run (like i in for loop).

Expert-Level Questions & Answers – Control Flow

No.	Question	Answer

1	◆w What is the output of
	for(int i = 0; i < 3; i++)
	if (i == 1) continue;
	System.out.print(i);?
		◆w Output: 02 – because 1 is skipped using continue.

2	◆w How do you break out of a nested loop in Java?	◆w Use a **labeled break**, e.g., break outer; to exit outer loop.
3	◆w Can we use a return inside a loop? What happens?	◆w Yes, it exits the whole method immediately, not just the loop.
4	◆w Can you use break and continue in the same loop?	◆w Yes, both can be used in different conditions within the same loop.
5	◆w What's the best way to skip processing for some elements in a loop?	◆w Use continue to skip the current iteration based on a condition.
6	◆w In which scenarios is do-while preferred over while?	◆w When you want the code block to run at least once (e.g., login retry, menu loops).
7	◆w How can we exit both inner and outer loops when a match is found?	◆w Use a **label** on the outer loop and break label; inside inner loop.

8	◆w What is the output of
	int i = 1;
	do {
	System.out.print(i);
	i--;
	}
	while(i > 0);?	